VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Různé algoritmy pro porovnávání textových dokumentů

Various algorithms for Text Documents Comparison

2014 Jiří Sýkora

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Sýkora**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Různé algoritmy pro porovnávání textových dokumentů
Various Algorithms for Text Documents Comparison

Zásady pro vypracování:

V době rozvoje informačních technologií je stále více řešena otázka podobnosti dokumentů. Algoritmy pro podobnost dokumentů se mohou uplatnit v široké aplikační oblasti - detekce plagiátů, detekce spamů, sumarizace a shlukování dokumentů, information retrieval, atd...
Cílem práce je tedy popsat existující algoritmy pro porovnání podobnosti dvou textových dokumentů, dále vybraný algoritmus naimplementovat a zhodnotit jeho výsledky.

1. Popište metody a algoritmy pro provnávání podobnosti textových dokumentů a popište klady a zápory mezi jednotlivými metodami a algoritmy.
2. Naimplementujte vybraný algoritmus pro porovnání podobnosti textových dokumentů.
3. Zhodnoťte výsledky naimplementovaných algoritmů.

Seznam doporučené odborné literatury:

[1] J.Pokorný, V.Snášel, D.Húsek.Dokumentografické informační systémy. Karolinum, Skriptum MFF UK Praha, 1998, ISBN 80-7184-764-X.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Berek**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24.4. 2014 .....

Rád bych na tomto místě poděkoval Ing. Petru Berkovi, za odborné, za pomoc a rady při zpracování této práce. Dále bych chtěl poděkovat své rodině a hlavně přítelkyni, za morální podporu a veškerou pomoc.

Abstrakt

V dnešní době, kdy se velmi rychle rozvíjí informační technologie, jsme nuceni stále častěji řešit otázku podobnosti dokumentů. V důsledku tohoto vznikla již spousta algoritmů, jež se zabývají právě touto problematikou. Mají široké využití, převážně při ověřování plagiátorství.

Tyto algoritmy se potýkají se dvěma problémy – efektivitou a výkonností. Program by měl být schopen objevit i jen malou část plagiovaného textu v celém dokumentu. Programy by ale měly zjistit plagiátorství i tehdy, pokud jsou frázová slova zaměněna za synonyma, věty jsou zpřeházené nebo dokonce upravené.

Toto jsou důvody, proč byla vymyšlena celá řada algoritmů pro porovnání souborů. Každý typ je založen na jiném systému, např. ověřuje počet výskytů slov nebo určuje podobnost pomocí vektorů. Takovéto algoritmy se tedy používají nejen pro ověřování plagiovaných prací, ale například booleovský a vektorový model jsou využívány i pro vyhledávače a vyhledávací systémy.

V této práci jsou popsány metody, kterými lze dva soubory porovnávnat. V teoretické části jsou uvedeny algoritmy, jež dané metody využívají. V praktické části je pak popsána implementace vybraných algoritmů, a to signaturní metody, Normalized Compression Distance a Fast Compression Distance. Ke konci této práce jsou pak naimplementované programy porovnány a zhodnoceny.

Klíčová slova: Java, plagiátorství, signaturní metody, signaturní soubory, signatury, booleovský model, vektorový model, Levenshteinova vzdálenost, Hammingova vzdálenost, Normalizovaná kompresní vzdálenost.

Abstract

Nowadays, when information technology is being quickly developed, we are forced to deal with questions about similarity of documents. As a result, a lot of algorithms which handle these problems have been created. They have a large use, especially in verification of plagiarism.

These are the reasons why a lot of algorithms for comparing files have been created. Each type is based on other system, e. g. it verifies the number of word occurences or it sets a similarity using vectors. These algorithms are used not only for verifying plagiarisms, but e. g. the Boolean and the vector models are used in the search systems.

In this thesis, the methods which can be used for file comparison are described. In the theoretic part the algorithms using these methods are shown. In the practical part an implementation of the chosen methods is showed – these include the signature method, the Normalized Compression Distance and the Fast Compression Distance. At the end of this thesis the implemented programs are compared and valorised.

Keywords: Java, plagiarisms, signature methods, signature files, signatures, bool model, vector model, Levenshtein distance, Hamming distance, Normalized Compression Distance.

Seznam použitých zkratek a symbolů

NCD	–	Normalized Compression Distance
FCD	–	Fast Compression Distance

Obsah

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílem mé bakalářské práce bylo popsat již existující metody a algoritmy pro porovnání podobnosti textových dokumentů. Dalším cílem bylo naiplmementovat vybraný algoritmus/algoritmy a popsat výsledky experimentů provedených na těchto algritmech.

V první části této práce se budu věnovat teoretickým informacím, které souvisejí s danou problematikou. V druhé části přiblížím metody, které byly implementovány a popíšu samotnou implementaci.

V této práci jsem se zaměřil na metody založené na vdálenosti, modelech a hashování. Teoretické informace jsou uvedeny v druhé kapitole. Rozebírá se zde Hammingova a Levenshteinova vzdálenost, NCD, FCD, signatury a booleovský a vektorový model.

Ve třetí kapitole se zmiňuji o těch metodách, které jsem implementoval. Jedná se o signaturní soubory, NCD a FCD.

Ve čtvrté kapitole se věnuji popisu datasetu a výsledkům z experimentů na daných metodách.

2 Metody pro porovnávání textových souborů

2.1 Metody založené na vzdálenosti

Tyto metody využívají vzdálenosti dvou řetězců/souborů. Tato vzdálenost vyjadřuje rozdílnost reprezentovanou číslem. Podmínky podobnosti a vzdálenosti jsou navzájem opačné a každá podobnost může být převedena do vzdálenosti nebo naopak.[11]

2.1.1 Hammingova a Levenshteinova vzdálenost

Hammingova i Levenshteinova metoda pracují se vzdáleností dvou textových řetězců. Hammingova metoda předpokládá v ideálním případě dva stejně dlouhé řetězce a uvádí počet pozic, na kterých se tyto řetězce liší.

Levenshteinova metoda je o něco složitější. Jde o jeden z nejpoužívanějších algoritmů pro zjištění vzdálenosti textových řetězců. Vyjadřuje podobnost/rozdílnost dvou textů na základně počtu změn, které je potřeba udělat, abychom z jednoho řetězce dostali druhý.

Existují dvě hlaví varianty této metody - první, jednodušší a starší metoda je založená na počtu úprav, které je nutno v řetězci provézt, abychom získali dva stejné řetězce. Závisí tedy na počtu vložení, smazání nebo substituci. Ve své nejjednodušší podobě tento algoritmus známý již přes čtyřicet let spočítá počet vložení, nahrazení a smazání nutných k této transformaci [3].

Druhá metoda je poněkud chytřejší a počítá i se složitostí daných operací, což ji dělá realističtější, ale za to náročnější na výpočty.

Jednou z možností je měření vzdálenosti na úrovni znaků, což není pro účely detekce plagiátů vhodné provádět. U větších dokumentů je vhodnější využít asociování slova do jednobajtového kódu a následně se v rámci věty porovnávají tyto kódy. Místo se znaky tak pracujeme s celými slovy. Jednobajtový kód je schopen pokrýt slova obou vět - nepředpokládá se, že by ve dvou větách bylo použito více než 256 slov.[3]

Tento způsob je citlivý i na drobné změny a jeho nespornou výhodou je, že je jako jeden z mála schopen pracovat na textech bez předchozího zpracování.

2.1.2 Normalized Compression Distance

Normalizovaná kompresní vzdálenost (NCD) vychází z metody pro výpočet normalizované informační vzdálenosti (NID). NID metoda je založena na Kolmogorově složitosti, která je ovšem nevypočitatelná a nejinak je to i s NID metodou. Pro metodu NID existuje vzorec:[5]

$NID_{(x,y)} = \frac{max\{K(x|y),K(y|x)\}}{max\{K(x),K(y)\}}$

kde K naznačuje rozdílnost na lexikografické úrovni. Tato metoda však není ani částečně vypočitatelná a proto vznikla metoda NCD, která už pracuje s binární délkou soborů a kompresorem.

Kolmogorova složitost je vyjádřením nákladnosti na výpočetní zdroje k jednoznačnému určení objektu - například textu. Složitost je vyjádřena jako délka nejkratšího popisu nad daným jazykem (programovací jazyk, česká abeceda...). Jestli program P vede

k získání slova s, pak P je jeho popisem. Délka popisu pak odpovídá délce bitů, které vyžaduje program P.

Kolmogorova funkce je nevypočitatelná z několika důvodů. Složitost jednoho programu závisí na zvoleném jazyku. Podmínkou navíc je, že daný program musí být nejkratší možný. V reálu ale neexistuje způsob, jak ověřit, že námi vytvořený algoritmus je opravdu ten nejkratší.

Kolmogorova složitost řetězce w s ohledem na jazyk L, psáno $K_L(w)$ je nejkratší program napsaný v jazyce L který vrátí w jako output. Podmíněná Kolmogorova složitost s ohledem na řetězec x, psáno $K_L(w|x)$ je délka nejkratšího programu, který, pokud dostane x jako vstup, vrátí w jako výstup.[9]

Budeme-li předpokládat, že U je spočitatelná funkce, která při vstupu jednoho binárního řetězce vrátí druhý, pak můžeme Kolmogorovu složitost vypočítat jako:[10]

$C_U(x|y) = min\{|p| \mid U(p, y) = x\}$

$C_U(x|y)$ je Kolmogorova složitost pro vstup y, kde výstupem bude x. $|p|$ udává délku programu.

Podobnost dvou textů/textových dokumentů lze vypočítat na základně NCD ze vzorce:[5]

$NCD_{(x,y)} = \frac{max\{C(xy)-C(x),C(yx)-C(y)\}}{max\{C(x),C(y)\}}$

kde:

- C je kompresní algoritmus
- C(x), C(y) je velikost komprimovaného souboru x nebo y
- C(xy) je komprimovaná velikost spojení souboru x se souborem y
- C(yx) je komprimovaná velikost spojení souboru y se souborem x

Vstupem programu jsou dva řetězce. Ty se komprimují pomocí kompresní metody. Takovou metodou může být LZMA, PPMZ nebo metoda BZIP2. Získaná data se dosadí do vzorce, který vrací čísla od 0 do 1, kdy 0 znamená totožnost obou řetězců a 1 vyjadřuje, že jsou zcela odlišné. Kvalita této metody závisí i na kvalitě zvolené kompresní metody.

2.1.3 Fast Compression Distance

Tato metoda je založena na slovníkové metodě. Slovník je vytvářen na základě LZ komprese (Lempel-Ziv). Podobnost se pak vypočítá dle následujícího vzorce:[6]

$FCD_{(x,y)} = \frac{|D(x)|-|\cap(D(x),D(y))|}{|D(x)|}$

kde:

- D(x) je slovník ze vstupního souboru x
- D(y) je slovník ze vstupního souboru y
- $|\cap(D(x), D(y))|$ vyjadřuje velikost průniku obou slovníků

Při vydělení pak získáme rozdílnost, která je reprezentována čísly mezi 0 a 1 - kde 0 znamená, že jsou stejné, a 1 vyjadřuje odlišnost.

2.2 Metody založené na modelech

Existují modely, které se zaměřují na různé typy porovnávání dokumentů - mohou pouze rozlišovat přítomnost/nepřítomnost v souboru, mohou rozlišovat i frekvenci výskytu určitých slov nebo mohou rozlišovat i pozice, na kterých se vyskytují. Takovéto systémy nacházejí využití při rychlém vyhledávání - například v knihovnách. Díky tomu se ale také dají efektivně využít při porovnávání dvou sobourů - vyhledávání slov z jehodo souboru v druhém.

2.2.1 Booleovský model

Booleovský model je nejstarším modelem textové databáze. Dokumenty jsou reprezentovány pomocí množin termů a dotazováním, které vychází z vyhodnocováním Booleovských výrazů. Booleovský výraz vrací hodnoty 0 nebo 1 (true nebo false) a je konstruován pomocí závorek a logických spojek.

Booleovský model může sloužit k porovnávání celých dokumentů, kdy jeden ze souborů tvoří dotaz a v dalším se hledají dané termy z dotazu. Termy jsou spojeny pomocí operátoru OR, čím větší shoda termů nastane, tím větší je podobnost souborů.

Dotaz je obecně vlastně posloupnost termů spojených operátory OR, AND, NOT a závorkami. Odpovědí na dotaz jsou dokumenty, jejichž popis splňuje zadanou podmínku. Závorky zpřehledňují syntaxi, často ovšem nejsou nutné. Některé vyhledávající stroje však důsledné závorkování vyžadují. Např: $A \wedge B \wedge C$ je třeba zapsat jako $((A \wedge B) \wedge C)$. Při dotazu s disjunkcí se vyberou dokumenty obsahující jeden z termů i dokumenty se všemi uvedenými termy. Všechny dokumenty na výstupu jsou systémem chápány jako stejně dobré. Pro uživatele by ale bylo nejlepší, kdyby byly dokumenty setříděné tak, aby na začátku byly "ty nejlepší".

Správně formulovaný Booleovský dotaz je schopný vracet poměrně přesné výsledky. Je jednoduše implementovatelný a časově efektivní.

2.2.2 Vektorový model

Vektorový model je přibližně o 20 let mladší než Booleovský model a můžeme o něm říct, že je i poměrně složitější. Zvažuje totiž i fakt, jak často se hledané slovo v dokumentu vyskytuje. Dokumenty i dotazy jsou v n-rozměrném prostoru (n je počet všech slov) a jsou reprezentovány vektory. Dokumenty podobné dotazu mají vektor podobného směru jako vektor dotazu. Tato podobnost se zjistí skalárním součinem. Předtím se ještě vektory znormalizují na jednotkovou délku, aby delší dokumenty nebyly zvýhodněny.

Definice 2.1 *Skalární součin dvou vektorů a(a_1, a_2, ..., a_n) a b(b_1, b_2, ..., b_n) se rovná:*
$a * b = \sum_{k=1,\ldots,n}^{n} a_i b_i$

Pokud máme vektory, které nejsou normalizované, použijeme následující vzorec pro výpočet kosinovy míry:[1]

Definice 2.2 $Sim(Q, D_i) = \sum_{k=1,\ldots,n}(q_k * w_{ik}) / \sqrt{(\sum_{k=1,\ldots,n}(w_{ik})^2 * \sum_{k=1,\ldots,n}(q_k)^2)}$

Když si představíme, že pro indexaci všech dokumentů v databázi bylo použito celkem N termů $(T_1, T_2, \ldots, T_n)$, tak každý dokument (D_i) ze souboru dokumentů (D) je reprezentovaný vektorem

$D_i = (w_{i1}, w_{i2}, \ldots w_{in})$

kde w_{ij} náleží $< 0; 1 > n$ kde w_{ij} je váha náležící termu t_j v identifikaci dokumentu D_i.[1] Tato váha ohodnocuje důležitost jednotlivých termů pro identifikaci dokumentu podle obsahu. Váha, která se rovná nule, je nejméně důležitá, váha rovna 1 pak představuje nejvyšší důležitost. Soubor dokumentů D je ve vektorovém modelu popsán pomocí matice D, ve které i-tý řádek odpovídá i tému dokumentu a j-tý sloupec j-tému termu z množiny T. Dotaz Q je potom možné vyjádřit jako n-místný vektor vah

$Q = (q_1, q_2, \ldots, q_n)$, kde $q_j \in < 0; 1 >$.[1]

Na základě dotazu Q jsme pak schopni pro každý dokument D_i spočítat koeficient podobnosti (angl.. similarity). Koeficient podobnosti si můžeme představit jako „vzdálenost“ vektoru dokumentu od vektoru dotazu v prostoru $< 0; 1 > n$. Z toho vyplývá, že čím jsou si blíže, tím jsou si podobnější.

Vyhodnocení dotazu ve vektorovém modelu je vlastně vybírání záznamů a předkládání je uživateli podle snižujícího se SIM – uživatel nejdříve dostane ty „nejlepší“ záznamy. Místo upřesňování dotazu pomocí dalších booleovských operátorů, mění se váhy termů v dotazu, případně se i vylaďují váhy termů v matici D.

Všimněme si, že když se všechny nenulové váhy ve vektorech nahradí jedničkami, vznikne nám Booleovský systém. Dotaz Q pak představuje disjunkci termů, koeficienty podobnosti udávají počet termů v D_i, které se shodují s některými termy v dotazu Q. Když chceme omezit množství výstupních dokumentů, máme dvě možnosti. Buď předepíšeme maximální přípustný počet záznamů, které chceme akceptovat, nebo stanovíme spodní prahovou hodnotu koeficientu podobnosti, kdy ještě dané dokumenty chápeme jako relevantní. Při ladění dotazu můžeme kromě přidávání a ubírání termů měnit i jejich hodnoty vah (významnosti).

Ve vektorovém modelu nelze rozlišit konjunktivní a disjunktivní dotaz. Špatně implementovaná je i operace NOT v pravém slova smyslu – tj. vyber dokumenty, které neobsahují... Řešit to můžeme pomocí rozšíření intervalu na $< -1; 1 >$. Záporné číslo pak snižuje podobnost dotazu vůči dokumentu.[1]

I když se vektorovým modelům vytýká, že předpokladem jejich dobré funkce je nezávislost termů (což je obvykle nesplněno), převyšují v provozu klasické Booleovské systémy. Někdy se uvádí zvýšení koeficientu přesnosti i o více než 100%. I přesto, že vektorový model je poněkud starší, práce s velkými textovými systémy ukázala, že se vyplatí zabývat se jím. Jejich funkce je velmi aktuální například co se týče Internetových vyhledávacích systémů – uživatel chce mít ten „nejlepší“ nalezený hned na prvních příčkách.

2.3 Metody založené na hashování

Hashovací funkce vemou řetězec libovolné délky a transformují jej na bytový řetězec s pevnou délkou. Jelikož je tento způsob kódování jednocestný, používá se například ke kódování hesel, tedy malých množství dat. Heslo, které napíše uživatel, se zpracuje pomocí některé hash funkce a uloží do databáze. Pokud se chce uživatel přihlásit, napíše své heslo, to se opět zpracuje pomocí hash funkce a porovná s hodnotou uloženou v databázi. Tento způsob je výhodný zejména proto, když dojde k neoprávněnému přístupu do databáze - všechna hesla jsou již pomocí hash funkce kódována, tedy je velice obtížné (i když ne úplně nemožné) zjistit původní heslo. [7]

Hash se vytváří pomocí hashovací funkce. Spousta programů využívá práci s hashovanými daty, protože umožňují jistou kompresi či převod na číselné symboly, s který se pracuje lépe než s řetězci. V takových případech se ovšem vyžaduje i převod z hashované hodnoty zpět na řetězec a proto se využívá zapisování dvojic klíč-hash.

Zápis lze provést pomocí pole, kde index je klíč a hodnota v poli na dané pozici je hash. To ovšem bývá velmi prostorově náročné a spousta místa se ani nevyužije. Na druhou stranu umožňuje rychlý přístup k datům. Dalším způsobem je použití seznamu - ten šetří místo - s každým novým záznamem jen přidá další hodnotu do seznamu. Na druhou stranu, vyhledávání v seznamu je sekvenční a tím velmi pomalé. Chceme-li jít střední cestou, můžeme využít ukládání pomocí binárního stromu. Stejně jako v seznamu v něm neexistují nevyužité klíče a stromová struktura oproti sekvenční značně usnadňuje vyhledávání.

Hashovací funkce má dobré i špatné vlastnosti. Výhodou je, že vždy pro stejný vstup vrátí i stejný výstup. Hashovací funkce ale bývají určitým způsobem omezené a tudíž nezaručují, že pro dva různé vstupy vrátí i dva různé výstupy. Využívá celého prostoru adres se stejnou pravděpodobností.[12]

Hashovací funkce se při porovnávání souborů používá k převední znaků nebo slov na číselné hodnoty. To usnadňuje práci a zrychluje běh programu.

2.3.1 Signaturní soubory

Metody signaturních souborů a ještě i některé další metody se používají nejen při výstavě textových databází, ale i v dalších oblastech. Mezi nejznámější systémy využívající signaturní soubory patří vyhledávací algoritmy pro více klíčů, programy pro kontrolu pravopisu nebo objektově orientované databáze. Signatury jsou zkomprimovaná data a signaturní soubory se používají pro vyhledávací dotazy, a to proto, aby se minimalizoval přístup k disku. Používání signatur je výhodné díky jejich binární jednoduchosti a malé velikosti při porovnání s původní velikostí nekomprimovaných dat (velikost signatury je 10-20%). U velkých souborů je rychlost za použití signatur výrazně pomalejší oproti jiným metodám.

Porovnávací proces začíná prohledáním vstupních souborů. Na každém slovu se vytvoří hashovací metodou signatura slova. Ta se pak skládá do signatury bloku nebo dokumentu. Signaturu slova lze vytvořit vícero způsoby. Jednou z možností je využití tripletů. O tvorbě se dočteme dále v této kapitole.

Definice 2.3 *Signaturou je d-bitový řetězec, $d > 0$, přirazený každému dokumentu v primárním souboru, resp. dotazu. Signatura dokumentu S_d vyhovuje signatuře dotazu S_q, pokud $S_d \wedge S_q = S_q$, tedy pokud všechny jedničkové bity v signatuře dotazu odpovídají jedničkovým bitům v signatuře textu. Předchozí podmínku lze efektivněji vyčíslit pokud se použije ekvivalentní dotaz $S_q \wedge \neg S_d = 0$.[1]*

Signatury můžeme vytvořit dvěma způsoby - zřetězením nebo spojením pomocí logické spojky OR. Oba tyto způsoby předpokládají existenci funkce, která zobrazuje termy na d-bitové řetězce (d je délka signatury). Výhodou vrstvených signatur je fakt, že dopředu známe jejich délku - víme tedy, kolik bude zabírat místa. Na druhou stranu, díky vrstvení dochází ke zkreslení. U vrstvení sice ke zkreslení nedochází, ale narůstá jejich velikost a tím vyžadují více paměti. Pokud vyjde signatura dokumentu se samými jedničkami, pak vyhovuje jakékoliv signatuře dotazu, což vede k nežádoucímu prohledávání textu. Abychom tomu zabránili, můžeme si soubor rozdělit do logických bloků. To se provádí pomocí rozdělení do bloků pevné délky nebo do bloků pevné váhy. Máme-li blok pevné váhy, pak dvě signatury například obsahují přibližně stejný počet termů nebo může být pravidlem váhy stejný počet nul a jedniček.[1]

Dalším aspektem, který hraje roli v přesnosti signaturních souborů, je velikost signatury a počet jedniček v ní. Vzhledem k tomu, že rozdílnost je určena rozdílným umístěním bitu s hodnout 1, je optimální, pokud má signatura stejný počet nul i jedniček.

Průměrný počet záznamů chybného výběru vyčteme ze vzorce:

$$P_f * m = F$$

- F je počet chybných výběrů
- m je počet souborů
- P_f je pravděpodobnost chybného výběru

Pro vrstvené kódování odvodil Roberts formule pro optimální délku signatur a váhu binárních reprezentací participujících termů. Nechť P(t) je pravděpodobnost, že signatura, která vznikla vrstvením n binárních reprezentací, obsahuje '1' na t daných pozicích. Roberts zdůvodnil, že:[1]

$$P(t) \sim [1 - (1 - \tfrac{k}{d})^n]^t$$

Pak:

$$P(1) = 1 - (1 - k/d)^n$$

- P(1) je pravděpodobnost, že signatura obsahuje 1 na t jedniček
- k je počet jedniček v signatuře
- d je velikost signatury
- n je počet termů v dokumentu

Výhody a nevýhody jsou použity z literatury [1].
Mezi výhody používání souborů signatur patří:

- Velikost souboru signatur je malá, je ji možné řídit.
- Cena údržby je malá, jde o jednoduchou oranizaci souboru.
- Signatury se generují jednoduše, cena vložení nové signatury je malá.
- Pomocné informace o souboru (signatury) nejsou rozptýleny do seznamů souřadnic jako u invertovaných souborů.
- Vrstvené kódování je vhodné zvláště pro konjunktivní dotazy s více termy. Zatímco cena dotazu se s počtem termů invertovaného souboru zvyšuje, u soubor signatur se snižuje.

Signaturní soubory mají také jisté nevýhody:

- Vyhlédávání je pomalé, je-li soubor signatur organizován jako sekvenční soubor, je třeba s dotazem porovnávat každou signaturu.
- Chybné výběry jsou drahé a obtížně se odstraňují.
- Informace o četnosti termů či váhy se obtížně do signatur kódují.
- Jiné než konjunktivní dotazy se implementují se signaturami obtížně (disjunkce, vzdálenosti, synonyma apod.)

Samotnou signaturu lze vytvořit více způsoby. Jedna metoda je založená na vektoru výskytů termů. Každému termu je přiřazen právě jeden bit. Signatura je tvořena vrstveným kódováním, kdy se jednotlivé signatury termů spojují pomocí spojky OR. V tabulce 1 je uveden příklad tvorby signatury souboru.

slovo	binární reprezentace
jablko	000 000 000 001
hruška	000 000 000 010
švestka	000 000 000 100
švestka	000 000 001 000
Signatura D	000 000 001 111

Tabulka 1: Binární reprezentace slov

Pomocí této metody jsme schopni dostat velice přesné výsledky, avšak její nevýhodou při velkém počtu termů je velmi dlouhá signatura. Protože je délka signatury proměnlivá pro každý dokument, musíme dopředu znát počet termů, který musí zůstat neměnný.

Další metodou je rozložení textu na bloky podle pevné délky. Každý blok obsahuje n termů. Každé slovo má vlastní binární reprezentaci dlouhou d bitů. Tato délka se nemění.

Z těchto d bitů právě k bitů nabývá hodnoty 1. Poté aplikací logického operátoru OR získáme signaturu bloku. Pro příklad vyhledávání pomocí této metody mějme dokument D obsahující termy: jablko, hruška, švestka, broskev, třešeň, meruňka. Dokument si rozdělíme do dvou bloků B, kde blok bude mít velikost $n = 3$, velikost signatury $d = 12$ a váha binární reprezentace $k = 3$. První blok je uveden v tabulce 2, druhý blok v tabulce 3.

jablko	000 000 000 111
hruška	000 100 000 011
švestka	100 010 010 000
Signatura B_1	100 110 010 111

Tabulka 2: První blok

broskev	011 001 000 000
třešeň	000 110 000 001
meruňka	010 110 000 000
Signatura B_2	011 111 000 001

Tabulka 3: Druhý blok

V tomto dokumentu chceme vyhledat term broskev. Signatura dotazu odpovídá binární reprezentaci broskve: 011001000000. Na signaturu bloku a dotazu aplikujeme logickou operaci AND.

Signatura B_1	100 110 010 111
Signatura dotazu	011 001 000 000
Výsledná signatura	000 000 000 000

Tabulka 4: Dotaz na první blok

Z tabulky 4 vyplývá, že hledaný term se v prvním bloku nenachází. Proto provedeme hledání i v druhém bloku.

Signatura B_2	011 111 000 001
Signatura dotazu	011 001 000 000
Výsledná signatura	011 001 000 000

Tabulka 5: Dotaz na druhý blok

Z tabulky 5 vidíme, že signatura dotazu odpovídá výsledné signatuře, což znamená, že hledaný term byl nalezen.

Jinou metodou pro tvorbu signatur je tvorba pomocí tripletů. Termy se v podstatě rozdělí na všechny po sobě jdoucí posloupnosi o délce tří znaků. Tyto vzniklé triplety jsou použity k vytvoření signatury dokumentu. Učitou nevýhodou této metody je zvýšení chybné kvalifikace. Jestliže máme stejný triplet z různých slov, dojde k vygenerování stejné hodnoty a tím ke kolizi.[4]

3 Vybrané metody pro porovnávání textových souborů

3.1 Signaturní soubory

Pro vytvoření signatury dokumentu byla použita metoda pracující s triplety. Nejprve načítáme slovo po slovu. Slova s délkou menší než tři znaky vyfiltrujeme. Každé slovo je pak rozděleno na triplety a každému trileptu je přiřazeno číslo od 0 do N. Délku signatury (N), jsem vypočetl ze vzorce uvedeného v kapitole 2.3.1. Byla zjištěna optimální délka pro tento dataset, a to 4947. Toto číslo jsem použil v implementaci. Například máme-li slovo abeceda, číselnou hodnotu přiřadíme tripletům abe - bec - ece- ced - eda.

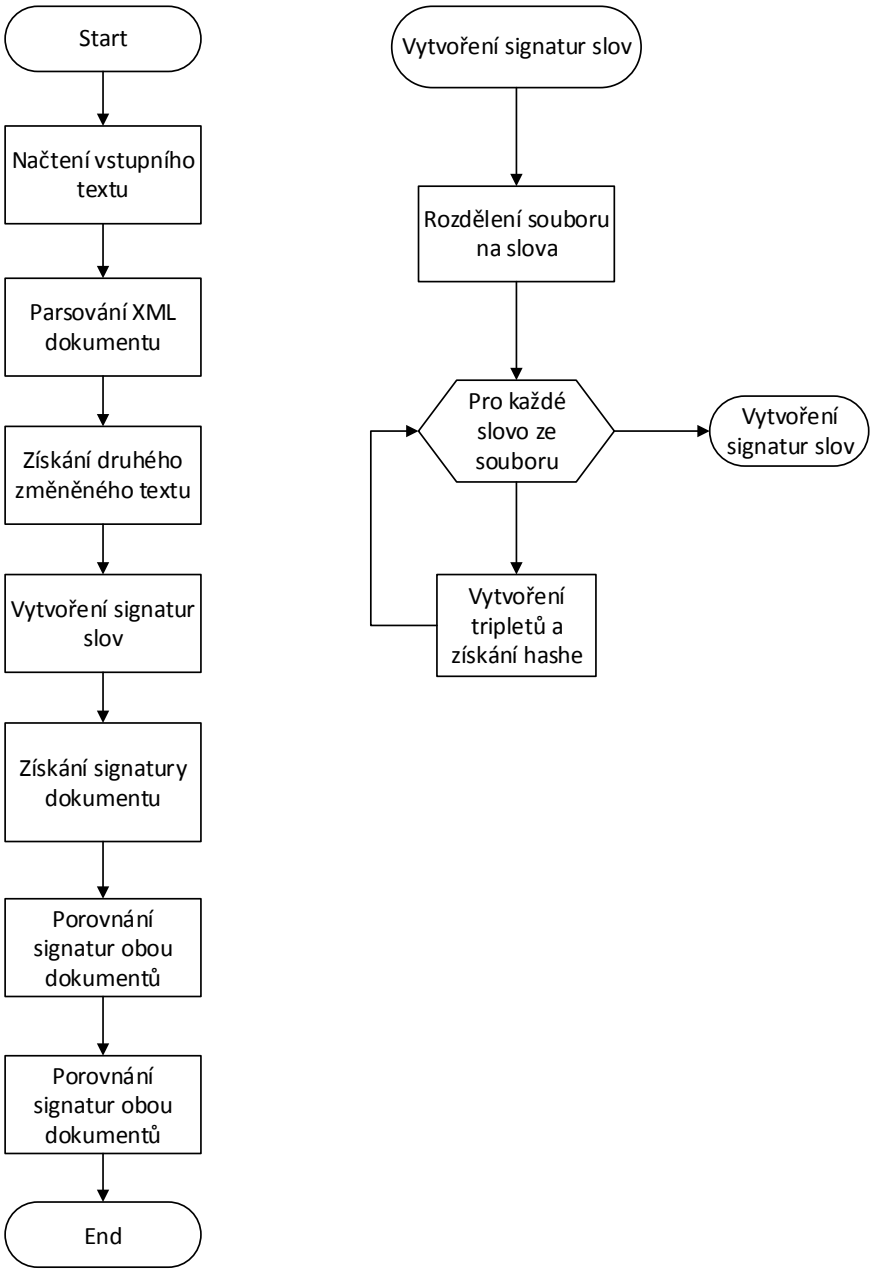
Hash hodnoty jednotlivých tripletů jsou řešeny následujícím kódem, který vychází z Knuthova hashovacího algoritmu:

```
private int hashTriplet(char[] triplet ){
        int hash = 0;
        for (int i = 0; i < 3; i++){
            hash = (64 * hash + triplet [i]) % N;
        }

        return hash;
    }
```

Výpis 1: Program v jazyce Java pro popis hashování tripletů

Číselné hodnoty vygenerované na základně tripletů nám říkají, na jaké pozici ve výsledné signatuře slova se bude nacházet jednička. Při zvolení nižšího čísla N dochází k častým konfliktům v signaturách slov a snižuje se tím přesnost porovnávání. Avšak zvolením příliš vysokého čísla N rostou nároky na pamět. Jakmile získáme signaturu slova, vložíme ji do seznamu signatur. V případě, že už se v tomto seznamu vyskytuje, znovu ji nevkládáme. Signturu dokumentu získáme tak, že aplikujeme operaci OR na jendotlivé signatury slov v seznamu. Celý proces je totožný i pro druhý dokument. Výslednou podobnost dvou dokumentů získáme porovnáním jejich signatur. Čím více se signatury liší, tím rozdílnější dané soubory jsou.



Obrázek 1: Vývojový diagram Signaturní metody

3.2 Normalized Compression Distance

Vstupem jsou dva textové soubory. V programu pracujeme s oběma texty a jejich konkatenací. Jednotlivé řetězce pak zkomprimujeme pomocí komprimační metody. V našem případě jsme využili knihovny GZIP. Je založena na algoritmu DEFLATE, který je kombinací LZ77 a Huffmanova kódování. GZIP metoda je oblíbená díky své spolehlivosti a je vcelku populární i na Internetu.

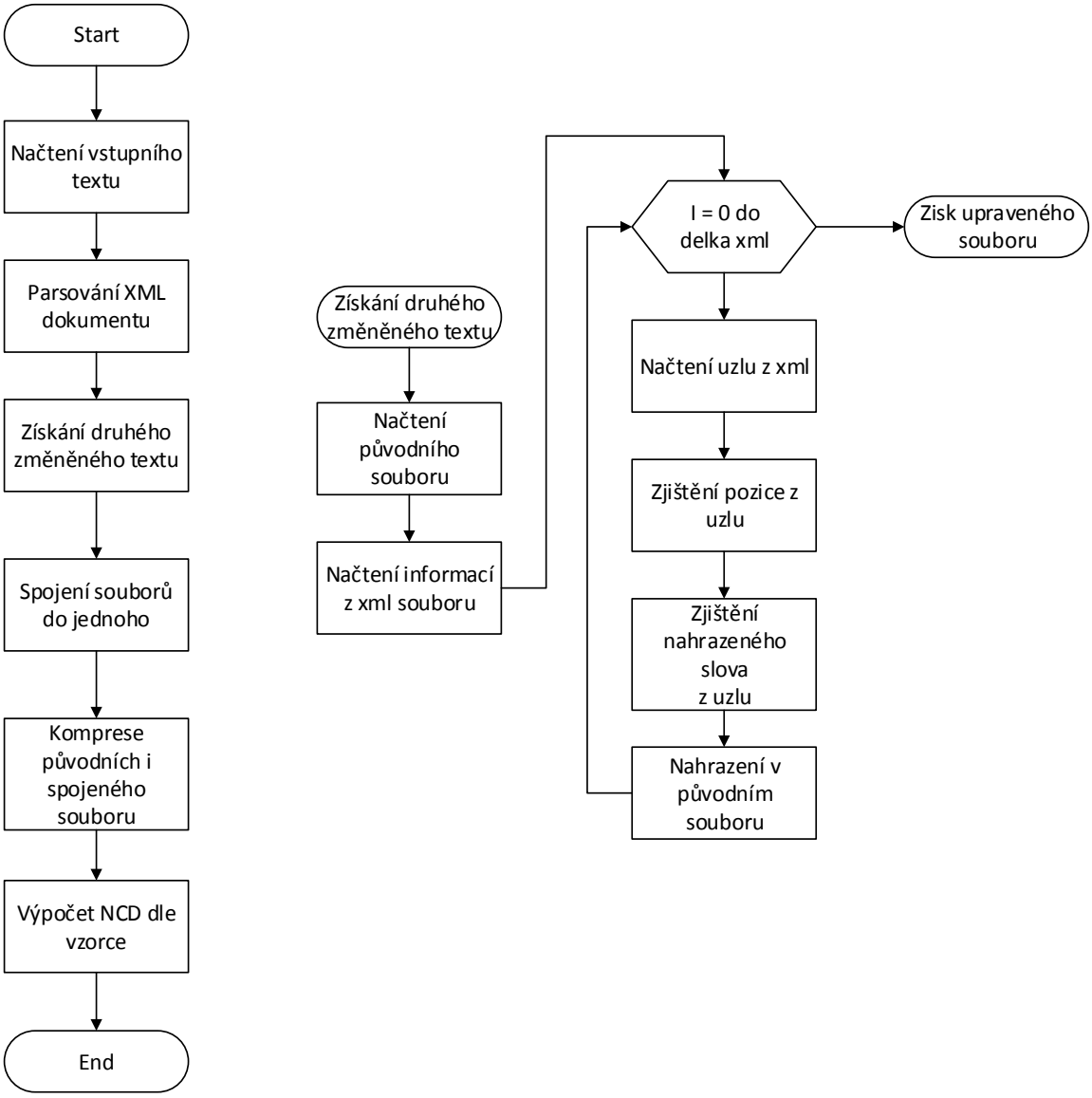
```
public static String compress(String str) throws IOException {
        if (str == null || str.length() == 0) {
            return str;
        }

        ByteArrayOutputStream out = new ByteArrayOutputStream();
        try (GZIPOutputStream gzip = new GZIPOutputStream(out)) {
            gzip.write(str.getBytes());
        }
        String outStr = out.toString("ISO-8859-1");

        return outStr;
    }
```

Výpis 2: Program v jazyce Java pro popis kompresního algoritmu.

Následně pomocí knihovny Math spočítáme podobnost ze vzorce, který je uveden v kapitole 3.2. Vzorec vrací čísla od 0 do 1, kdy 0 znamená totožnost obou řetězců a 1 vyjadřuje, že jsou zcela odlišné.



Obrázek 2: Vývojový diagram NCD

3.3 Fast Compression Distance

Vstupem jsou dva textové soubory. V programu jsou reprezentovány řetězci. Každý řetězec pak zkomprimujeme pomocí metody LZW, což je slovníková metoda. Pro každý soubor je vytvořen slovník. Následující kód uzkazuje vytváření slovníku.

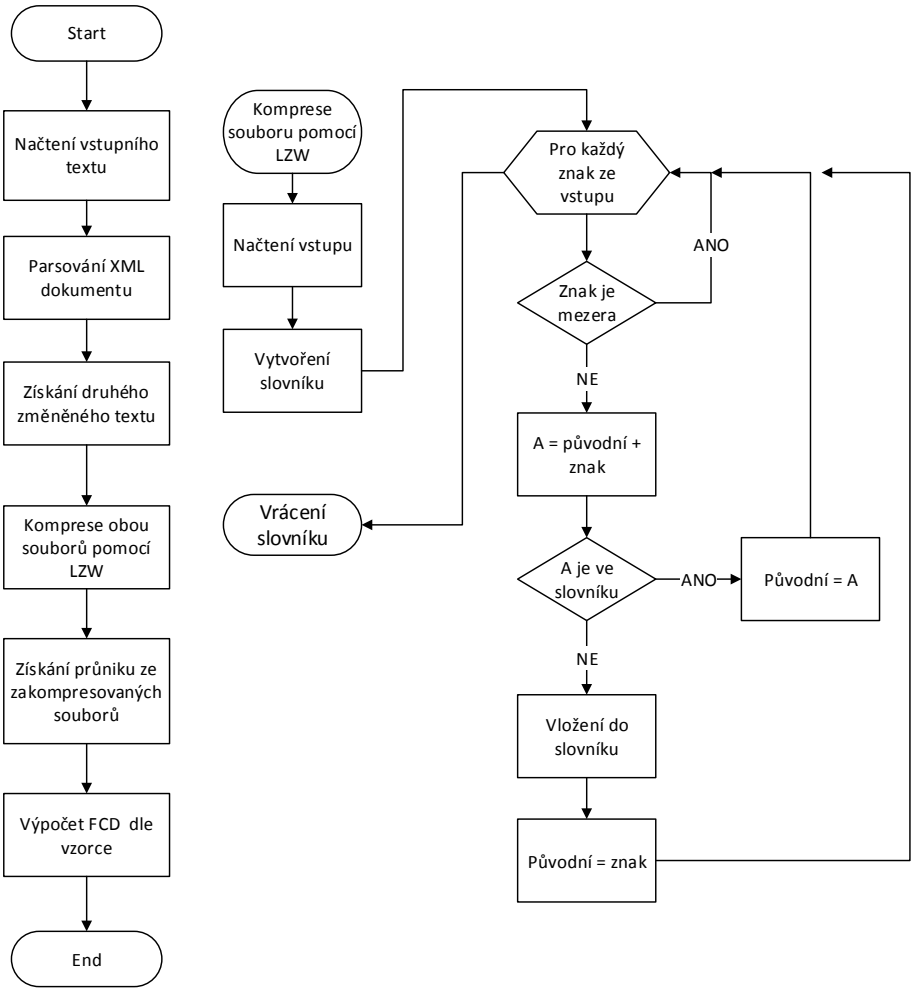
```
public static Map<String, Integer> compress(String s) {
    int dictSize = 0;
    Map<String, Integer> dictionary = new HashMap<>();

    String w = "";
    for (char c : s.toCharArray()) {
        if (c == '␣') {
            continue;
        }

        String wc = new StringBuilder(w).append(c).toString();
        if (dictionary.containsKey(wc)) {
            w = wc;
        } else {
            dictionary.put(wc, dictSize++);
            w = new StringBuilder("").append(c).toString();
        }
    return dictionary;
}
```

Výpis 3: Program v jazyce Java pro popis vytváření slovníku LZW metodou.

Tyto slovníky jsou dále převedeny do TreeMap. TreeMap je založena na stromové struktuře. Slovník je pomocí této struktury seřazen na základě klíče. Dále si získáme velikosti slovníků a velikost jejich průniku. Tyto získáné hodnoty dosadíme do vzorce uvedeného v kapitole 2.1.3.



Obrázek 3: Vývojový diagram FCD

4 Popis datasetu a experimentů vybraných metod

4.1 Dataset

Jako dataset jsem použil Federalist Papers. Federalist Papers je soubor 85 článků a esejí napsaných Alexandrem Hamiltnem, Jamesem Madisonem a Johnem Jayem na podporu schválení Ústavy Spojených Států. 77 článků bylo publikováno v The Independetn Journal a The New Yourk Packet v letech 1787-1788. Zbytek článků byl vydán až později. Původní název byl The Federalist, označení The Federalist Papers se vžilo až ve dvacátém století. V době zveřejnění článků bylo autorství přísně tajné, časem se autorství odkrylo.

The Federalist Papers se skládá z 85 článků. V datasetu využívám těchto článků. Ke každému z nich existuje 99 xml souborů. Každý xml soubor se skládá z hlavičky, ve které je řečeno, kolikaprocentní změnu tento soubor obsahuje. Následně je v xml vždy uvedena pozice, na které došlo ke změně, a slovo, za které se má to původní zaměnit. Nové slovo je generováno z množiny slov, která se jinak v dokumentu nevyskytuje.

```
<?xml version="1.0" encoding="utf−8"?>
<generator replacepercent="1,0625">
  <word>
    <position>833</position>
    <original>the</original>
    <replacement>resistance</replacement>
  </word>
  <word>
    <position>693</position>
    <original>spirit</original>
    <replacement>reducible</replacement>
  </word>
  <word>
    <position>1269</position>
    <original>they</original>
    <replacement>invalidates</replacement>
  </word>
</generator>
```

Výpis 4: Ukázka XML souboru

XML nám udává pozici slova (position), orinální slovo (original) a slovo, kterým originální slovo bude nahrazeno (replacement).

4.2 Experimenty a zhodnocení naimplementovaných metod

Každá naimplemenovaná metoda tiskne výsledky do souboru .csv, které jsou pak v této kapitole dále zpracovávány.

4.2.1 Výsledky Signaturní metody

Procentuální změna	Podobnost
1%	2,38%
10%	19,75%
20%	34,3%
30%	45,06%
40%	54,01%
50%	61,6%
60%	68,39%
70%	74,13%
80%	79,5%
90%	83,11%
99%	87,22%

Tabulka 6: Podobnost u Signaturní metody

Tabulka 6 je vytvořena na základě výsledků vygenerovaných programem do csv souboru. Tato tabulka nám ukazuje reálnou procentuální změnu, která byla provedena na originálním dokumentu (levý sloupec). V pravém sloupci jsou uvedeny průměrné hodnoty podobnosti, získané pomocí signaturní metody, nad celým datasetem. Pro všech 86 souborů byla vypočtena podobnost na souborech o odpovídající změně a tyto hodnoty byly následně zprůměrovány. Pro větší přehlednost jsem zde nezvolil krok po jednom procentu, ale po deseti procentech.

Při optimálně zvolené délce signatury N, dostáváme poměrně přesné výsledky. Průměrná odchylka je 8,89%, největší odchylka je při 30% změně, kdy nám vychází rozdílnost 45,06% - což je o 15,06% více. Ze všech testovaných metod nám signaturní metoda vrací nejpřesnější výsledky. Nejpřesnějších výsledků dosahuje tato metoda při rozdílnosti souborů nad 50%.

4.2.2 Výsledky Normalized Compression Distance metody

Procentuální změna	Výsledná podobnost
1%	7,89%
10%	32,35%
20%	51,54%
30%	64,72%
40%	74,2%
50%	81,07%
60%	86,26%
70%	89,91%
80%	92,63%
90%	94,66%
99%	96,07%

Tabulka 7: Podobnost u NCD

Struktura tabulky je stejná, jako u výše uvedené signaturní metody.

Průměrná odchylka je 20,65%, což je mnohem větší odchylka než u předchozích signaturních souborů. Největší odchylka je opět při 30% změně, kdy nám vychází rozdílnost 64,72% - což je o 34,72% více.

4.2.3 Výsledky Fast Compression Distance metody

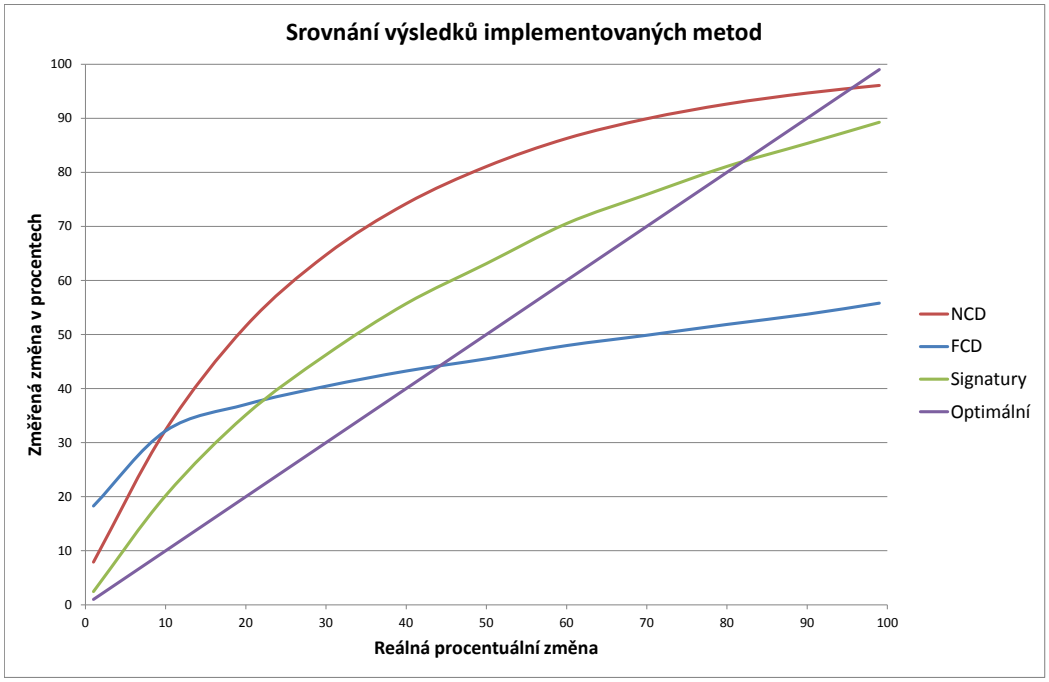
Procentuální změna	Výsledná podobnost
1%	18,28%
10%	32,17%
20%	37,06%
30%	40,43%
40%	43,24%
50%	45,5%
60%	47,95%
70%	49,86%
80%	51,86%
90%	53,75%
99%	55,8%

Tabulka 8: Podobnost u FCD

Struktura tabulky je stejná, jako u výše uvedených metod.

Průměrná odchylka je 23,6%, což je větší odchylka než u obou předchozích porovnávacích metod. Největší odchylka je při 99% změně, kdy nám vychází rozdílnost 55,8% - což je o 43,2% více. Z hodnot vidíme, že roste do 20% změny, pak ale nastává zlom a procentuální přírustek už není tak velký.

4.2.4 Porovnání výsledků



Obrázek 4: Srovnání výsledků

Na grafu 4 lze vidět porovnání průměrných výsledků naimplementovaných metod a také optimální křivka, ke které by se metody měly blížit.

Z grafu lze vyčíst, že nejvíce se k optimální křivce blíží křivka signaturní metody. Po celý běh programu vykazuje jen mírně větší hodnoty, než jsou na optimální křivce, při rozdílnosti nad 80% pak optimální křivku protíná.

Metoda NCD také kopíruje běh optimální křivky, i když s poněkud větší odchylkou, než tomu bylo u signaturní metody. Při nejvyšší rozdílnosti souborů dosahuje nejhorších výsledků FCD metoda, kde je odchylka až 40%.

5 Závěr

Cílem této práce bylo popsat již existující metody a algoritmy pro porovnání podobnosti textových dokumentů. Dalším cílem bylo naiplmementovat vybrané algoritmy a popsat výsledky experimentů provedených na těchto algritmech.

V první části této práce jsem se věnoval teoretickým informacím, které souvisejí s danou problematikou. V druhé části jsem pak přiblížil metody, které byly implementovány a popsal samotnou implementaci. Nakonec jsem porovnal výsledky z experimentů.

Jako testovací dataset jsem zvolil The Federalist Papers. Pro načítání změn v orignálních souborech jsem využil parsování xml dokumentů, udávajích změny provedené na původních dokumentech.

Pro implementaci jsem zvolil signaturní metodu, což je metoda zastupující metody založené na hashování, a další dva algoritmy, vycházející z metod založených na vzdálenosti - NCD a FCD. Pro implementaci jsem zvolil programovací jazyk Java.

Pro implementaci signaturní metody jsem využil tripletů a hashovací funkce založené na Knuthově hashovacím algoritmu. Délku signatury jsem vypočítal ze vzorce v závislostni na použitém datasetu.

V algoritmu NCD jsem použil GZIP metodu vycházející z knihovny GZIP, kterou Java obsahuje. Jako kompresní metodu ve FCD jsem použil původně LZ78, která je založena na stromech, ale nakonec jsem se z důvodu nepřesných výsledků rozhodl použít metodu LZW, která pro svou implementaci využívá slovníků. Tyto slovníky jsem pak seřadil pomocí převodu do TreeMap.

Z experimentů jsem zjistil, že nejpřesnější metodou z testovaných je metoda využívající signatur. Maximální odchylka byla pouhých 15%, průměrná pak jen necelých 9%. Oproti tomu nejhůře na tom byl algoritmus popisující FCD, s průměrnou odchylkou 23% a maximální odchylkou celých 43%.

Téma této bakalářské práce se mi zdálo velmi zajímavé. Přimělo mě nastudovat si různé metody a vyzkoušel jsem si jejich implementaci i v praxi. Myslím, že na tomto poli je stále co zlepšovat a určitě existuje spousta způsobů, jak existující algoritmy nebo jejich části zefektivnit.

Tato práce byla napsána v typografickém systému LaTeX.[2]

6 Reference

[1] J. Pokorný, V. Snášel, D. Húsek, *Dokumentografické informační systémy*, 1. vyd. Praha: Karolinum, 1998. ISBN 80-7184-764-x.

[2] RYBIČKA, Jiří, *Latex pro začátečníky*, 3. vyd. Brno: Konvoj, 2003, 238 s. ISBN 80-730-2049-1.

[3] J. Přibil, *Efektivní metody detecke plagiátů v rozsáhlých dokumentových skladech*, Vysoká škola ekonomická v Praze, 2010. Doktorská dizertační práce. Fakulta managementu v Jindřichově Hradci. Dostupné na internetu: `http://www.fm.vse.cz/wp-content/uploads/2011/04/Pribil_Jiri_DP.pdf`

[4] Ing. Zdeněk Korčák, *Signaturní soubory se signaturami proměnné délky*, Vysoké učení v Brně, 2002. Doktorská dizertační práce. Fakulta informačních technologií. Dostupné na internetu: `http://www.fit.vutbr.cz/~zendulka/theses/zkorcak.pdf`

[5] Ana Granados Fontecha, *ANALYSIS AND STUDY ON TEXT REPRESENTATION TO IMPROVE THE ACCURACY OF THE NORMALIZED COMPRESSION DISTANCE*, Universidad Autónoma de Madrid, 2012. Doktorská dizertační práce. Escuela Politécnica Superior. Dostupné na internetu: `http://arxiv.org/pdf/1205.6376v1.pdf`

[6] Daniele Cerra* and Mihai Datcu, *A Fast Compression-based Similarity Measure with Applications to Content-based Image Retrieval*, German Aerospace Center (DLR), Earth Observation Center (EOC), 2012. Dostupné na internetu: `http://arxiv.org/ftp/arxiv/papers/1210/1210.0758.pdf`

[7] Marek Běhálek, *Kódování a hashování*, VŠB-TU Ostrava, 2007. Fakulta elektrotechniky a informatiky. Dostupné na internetu: `http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch09s01.html`

[8] KUN, Jeremy, *Kolmogorov Complexity - A Primer Math ∩ Programming*, [online]. 2012. [cit. 2014-04-29]. Dostupné na internetu: `http://jeremykun.com/2012/04/21/kolmogorov-complexity-a-primer/`

[9] Lance Fortnow, *Kolmogorov Complexity*, [online]. 2012., [cit. 2014-04-29]., Dostupné na internetu: `http://jeremykun.com/2012/04/21/kolmogorov-complexity-a-primer/`

[10] Nikolay Vereshchagin, *Randomized communication complexity of approximating Kolmogorov complexity*, Higher School of Economics, Yandex, Moscow State University. 2013.

[11] *Analysis of genomic and proteomic data*, [online]., [cit. 2014-04-29]., Dostupné na internetu: `http://telemedicina.med.muni.cz/genomic-proteomic-analysis`

[12] *Algoritmy.net,* [online]., [cit. 2014-04-24]., Dostupné na internetu: `http://www.algoritmy.net/article/32077/Hashovaci-tabulka`

7 Přílohy

DVD příloha obsahuje soubory:

federalistpaper - složka datasetu

FCD - Naimplementovaná FCD metoda

NCD - Naimplementovaná NCD metoda

Signatury - Naimplementovaná Signaturní metoda

vystupFCD.csv - Výsledky FCD metody nad celým datasetem

vystupNCD.csv Výsledky NCD metody nad celým datasetem

vystupSignatury.csv - Výsledky Signaturní metody nad celým datasetem